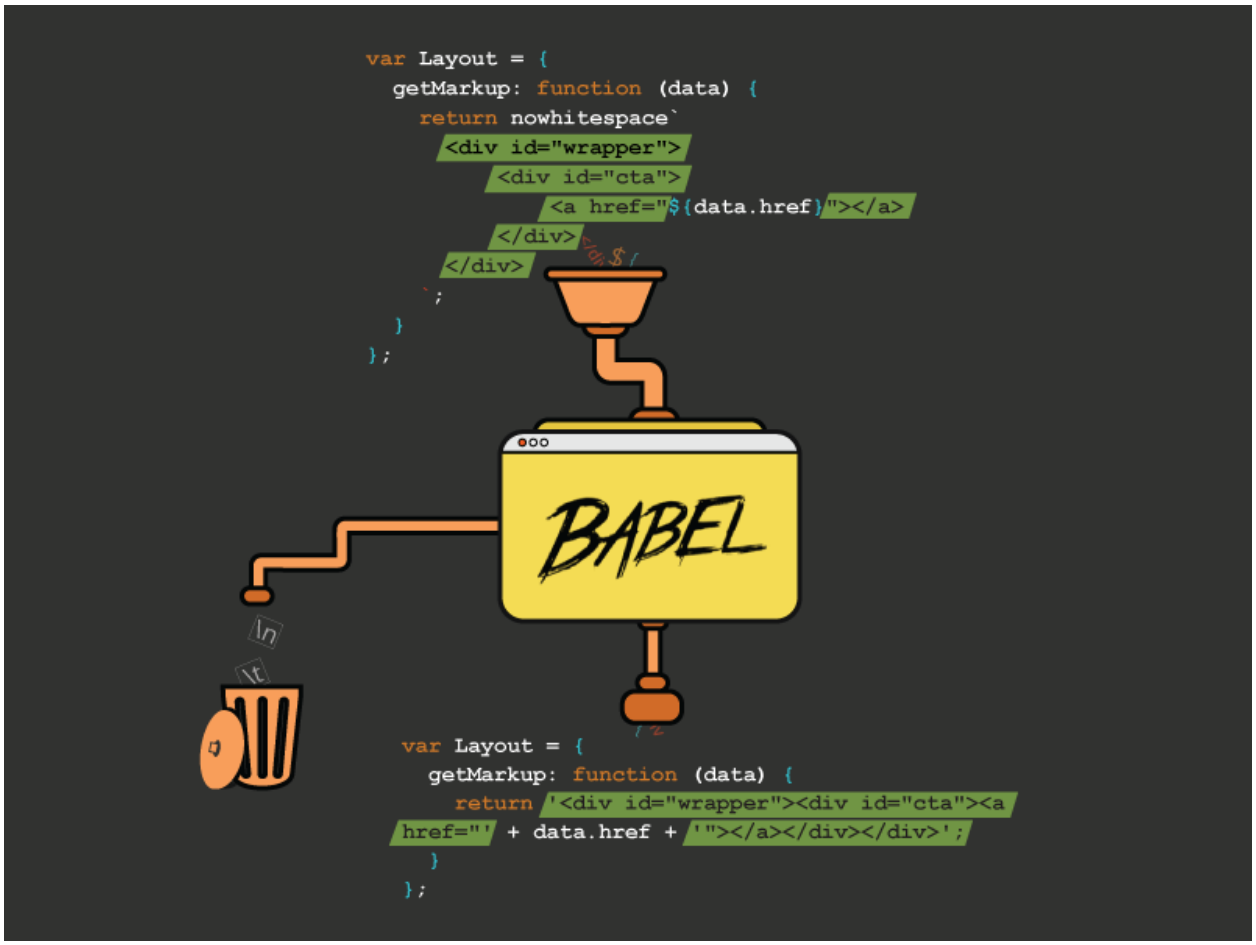
Date=4/11/2020
Lecture By=Manish Mahant
Subject ⇒Setting Up Bebel

IN PREVIOUS LECTURE **(QUICK RECAP) Date-3/11/2020**	**In Today's Lecture (Overview)**
What Is Javascript linters JSLint Example Getting Started with ESLint Installation and Usage Configuration ECMAScript 2015 - ES6 New Features in ES6 JavaScript let JavaScript const Arrow Functions Question For Self-Practice // CC For The Day	Babel is a JavaScript compiler Usage Guide Overview Core Library Configuration What Is Lodash In Javascript Installation Why Lodash? Questions For Self-practice // Assignment For The Day

Babel is a JavaScript compiler



Babel is a toolchain that is mainly used to convert ECMAScript 2015+ code into a backwards compatible version of JavaScript in current and older browsers or environments. Here are the main things Babel can do for you:

- Transform syntax
- Polyfill features that are missing in your target environment (through @babel/polyfill)
- Source code transformations (codemods)
- And more! (check out these videos for inspiration)

```
// Babel Input: ES2015 arrow function

[1, 2, 3].map((n) => n + 1);

// Babel Output: ES5 equivalent
```

```
[1, 2, 3].map(function(n) {

  return n + 1;

});
```

For an awesome tutorial on compilers, check out the-super-tiny-compiler, which also explains how Babel itself works on a high level.

Usage Guide

There are quite a few tools in the Babel toolchain that try to make it easy for you to use Babel whether you're an "end-user" or building an integration of Babel itself. This will be a quick introduction to those tools and you can read more about them in the "Usage" section of the docs.If you're using a framework, the work of configuring Babel might be different or actually already handled for you. Check out our interactive setup guide instead.

Overview

This guide will show you how to compile your JavaScript application code that uses ES2015+ syntax into code that works in current browsers. That will involve both transforming new syntax and polyfilling missing features.

The entire process to set this up involves:

1. Running these commands to install the packages:

npm install --save-dev @babel/core @babel/cli @babel/preset-env

2. npm install --save @babel/polyfill

Creating a config file named `babel.config.json` (requires `v7.8.0` and above) in the root of your project with this content:

```
{
    "presets": [
      [
        "@babel/env",
        {
          "targets": {
            "edge": "17",
            "firefox": "60",
            "chrome": "67",
            "safari": "11.1",
          },
          "useBuiltIns": "usage",
          "corejs": "3.6.5",
        }
      ]
    ]
  }
```

The browsers list above is just an arbitrary example. You will have to adapt it for the browsers you want to support.

```
const presets = [
    [
      "@babel/env",
      {
        targets: {
          edge: "17",
          firefox: "60",
          chrome: "67",
          safari: "11.1",
        },
        useBuiltIns: "usage",
        "corejs": "3.6.4",
      },
    ],
  ];
```

And running this command to compile all your code from the `src` directory to `lib`:

Core Library

The core functionality of Babel resides at the @babel/core module. After installing it:

`npm install --save-dev @babel/core`

you can `require` it directly in your JavaScript program and use it like this:

```
const babel = require("@babel/core");

babel.transform("code", optionsObject);
```

Now any arrow functions in our code will be transformed into ES5 compatible function expressions:

```
const fn = () => 1;

// converted to

var fn = function fn() {
  return 1;
};
```

Configuration

There are a few different ways to use configuration files depending on your needs. Be sure to read our in-depth guide on how to configure Babel for more information.

For now, let's create a file called `babel.config.json` (requires `v7.8.0` and above) with the following content:

```
{
    "presets": [
      [
      "@babel/env",
        {
          "targets": {
            "edge": "17",
            "firefox": "60",
            "chrome": "67",
            "safari": "11.1"
```

```
            }
          }
        ]
      ]
    }
```

What Is Lodash In Javascript

Lodash is a modern JavaScript utility library delivering modularity, performance & extras.

Installation

In a browser:

```
<script src="lodash.js"></script>
```

Using npm:

```
$ npm i -g npm
$ npm i --save lodash
```

In Node.js:

```
// Load the full build.
var _ = require('lodash');
// Load the core build.
var _ = require('lodash/core');
// Load the FP build for immutable auto-curried iteratee-first
data-last methods.
var fp = require('lodash/fp');

// Load method categories.
var array = require('lodash/array');
var object = require('lodash/fp/object');

// Cherry-pick methods for smaller browserify/rollup/webpack
bundles.
```

```
var at = require('lodash/at');
var curryN = require('lodash/fp/curryN');
```

Note:

Install n_ for Lodash use in the Node.js < 6 REPL.

Why Lodash?

Lodash makes JavaScript easier by taking the hassle out of working with arrays, numbers, objects, strings, etc.

Lodash's modular methods are great for:

- Iterating arrays, objects, & strings
- Manipulating & testing values
- Creating composite functions

Questions For Self-practice // Assignment For The Day

https://au-assignment.s3.amazonaws.com/Week_19_Day_3_Assignment-962441bf-adf8-466d-a8ae-98e27be547bc.pdf